Introduction to Programming for Biological Research
Today:

- Documenting your code
- Error handling
- Debugging
- Structured arrays
- Mathematical modeling
Error Handling

I HAVE NO IDEA

WHAT I'M DOING
Error Handling

• Document your code thoroughly

• Check for user errors and give helpful error messages
Documenting Your Code

We’ve already introduced you to the `help` command:

```
>> help plot
plot   Linear plot.
     plot(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
     then the vector is plotted versus the rows or columns of the matrix,
     whichever line up. If X is a scalar and Y is a vector, disconnected
     line objects are created and plotted as discrete points vertically at
     X.

     plot(Y) plots the columns of Y versus their index.
     If Y is complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).
     In all other uses of plot, the imaginary part is ignored.

Various line types, plot symbols and colors may be obtained with
plot(X,Y,S) where S is a character string made from one element
from any or all the following 3 columns:

<table>
<thead>
<tr>
<th>b</th>
<th>g</th>
<th>r</th>
<th>c</th>
<th>m</th>
<th>y</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>green</td>
<td>red</td>
<td>cyan</td>
<td>magenta</td>
<td>yellow</td>
<td>black</td>
</tr>
<tr>
<td>.</td>
<td>o</td>
<td>x</td>
<td>+</td>
<td>*</td>
<td>s</td>
<td></td>
</tr>
</tbody>
</table>
Documenting Your Code

You can write your own `help` documentation:

The first block of comments you make before the function declaration...

...is what’s shown when the user runs the `help` command.
Documenting Your Code

A README is another way to explain your code to the user:

```
Analysis code for 3D SIM images of FDAA labeling data

Georgia Squyres, Garner Lab, Harvard University
Written for MATLAB R2015a
Please report bugs or compatibility issues to squyres@g.harvard.edu

+++++++ 

ellipsoidFit.m: This function takes an image as an input, and returns ellipse fit parameters for that image, as well as the goodness-of-fit score.
Example: pr = ellipsoidFit(img)
pr is a 1x5 array, formatted as [x0, y0, major axis, minor axis, theta]
You can tweak some of the ellipse fitting parameters in the “user specified parameters” section of this code, at the top. However, I think my default settings for those parameters are pretty good, so you may not want to change them unless you are having trouble.

+++++++ 

analyzeFit.m: This function takes an image and ellipse fit parameters as inputs, and returns information about the closure, arc length, and intensity in the ellipse.
Example: [intensity, closure, arcLength] = analyzeFit(img,pr)
analyzeFit will also accept additional inputs to allow the user to choose a cutoff threshold. (If this is not included, it will identify a threshold by Otsu’s method)
Example: [intensity, closure, arcLength] = analyzeFit(img,pr,'thresh',0.01)
intensity is a scalar value indicating the sum of the intensity in the ellipse closure is a scalar value indicating the percentage of the ellipse labeled
```
Handling Errors

Suppose we have the following function:

```matlab
function list2 = myFun(list)
    for i = 3:5
        list(i) = list(i) + 2;
    end
end
```
Handling Errors

Suppose we have the following function:

```matlab
function list2 = myFun(list)
    for i = 3:5
        list(i) = list(i) + 2;
    end
end
```

What happens if we do this?
Handling Errors

Suppose we have the following function:

```matlab
function list2 = myFun(list)
    for i = 3:5
        list(i) = list(i) + 2;
    end
end
```

What happens if we do this?

![Command Window output](https://via.placeholder.com/150)
Handling Errors

Suppose we have the following function:

```matlab
function list2 = myFun(list)
    for i = 3:5
        list(i) = list(i) + 2;
    end
end
```

Now, what happens if we do this?

![Command Window](image)
Handling Errors

Suppose we have the following function:

```
function list2 = myFun(list)
    for i = 3:5
        list(i) = list(i) + 2;
    end
end
```

Now, what happens if we do this?
Handling Errors

This error has some helpful information for the programmer:

```
>> myFun([1 2 3 4])
Attempted to access list(5); index out of bounds because numel(list)=4.

Error in myFun (line 3)
    list(i) = list(i) + 2;
```

But we would like to give our users a more simple and helpful error message.
How Can We Check For Errors?

Add a conditional statement to this function that checks for the correct input:

```matlab
function list2 = myFun(list)
    for i = 3:5
        list(i) = list(i) + 2;
    end
end
```
How Can We Check For Errors?

Add a conditional statement to this function that checks for the correct input:

```matlab
function list2 = myFun(list)
    if length(list) >= 5
        for i = 3:5
            list(i) = list(i) + 2;
        end
    end
end
```
How Can We Check For Errors?

Add a conditional statement to this function that checks for the correct input:

Now, let's add an error message to the user:

```matlab
function list2 = myFun(list)
    if length(list) >= 5
        for i = 3:5
            list(i) = list(i) + 2;
        end
    else
        error('Your list is too short!')
    end
end
```
How Can We Check For Errors?

Now, when we run our modified code:

```
Command Window
>> myFun([1 2 3 4])
Error using myFun (line 7)
Your list is too short!
```

I have no idea what I'm doing.
How Can We Check For Errors?

Now, when we run our modified code:

```
Command Window

>> myFun([1 2 3 4])
Error using myFun (line 7)
Your list is too short!
```

There are other ways to handle errors, most notably with **try/catch blocks** (http://www.mathworks.com/help/matlab/ref/try.html)

However, writing conditional statements to catch errors is a good place to start!
Tools for Debugging

Suppose we have the following function:

```matlab
function list2 = myFun(list)
    for i = 3:5
        list(i) = list(i) + 2;
    end
end
```

Now, what happens if we do this?

```
>> myFun([1 2 3 4])
Attempted to access list(5); index out of bounds because numel(list)=4.
Error in myFun (line 3)
    list(i) = list(i) + 2;
```

Tools for Debugging

MATLAB also has a built-in debugger:
Tools for Debugging

Now, when I press Run, the program runs until the first break point and stops:

Variables in the function go to my workspace:

And I can interact with them at the command line:
Tools for Debugging

Now, when I press Run, the program runs until the first break point and stops:

Variables in the function go to my workspace:

And I can interact with them at the command line:

![Workspace Image]

![Command Window Image]

![Debugging Interface Image]
One Last Data Structure

We have already introduced you to variable types (double, logical, char)

We have also introduced you to data structures, ways of storing multiple pieces of information in an organized way: arrays, cell arrays

There is one other type of data structure you should know in MATLAB: the **structure array**.
Structure Arrays

Why use structure arrays?

1. They let you store your data in a very organized way that is easy for others to understand

2. Many MATLAB functions use structure arrays by default
Structure Arrays

The MATLAB function `dir` gives you a list of the files in the directory of your choice.

```
>> files = dir('/Users/georgiasquyres/Desktop/ImageData');
>> files

files =

99x1 struct array with fields:
    name
    date
    bytes
    isdir
    datenum
```

```
Structure Arrays

The MATLAB function `dir` gives you a list of the files in the directory of your choice.

```matlab
>> files = dir('/Users/georgiasquyres/Desktop/ImageData');
>> files

files =
    99x1 struct array with fields:
        name
        date
        bytes
        isdir
        datenum
```
Using Structure Arrays

Imagine we are tracking flies, like Matt does:

We have a lot of information (multiple variables) about each fly

We probably want to do this for multiple flies…
Using Structure Arrays

You can make a structure with `struct`:

```matlab
>> flies = struct('FlyName','fly1','xPos',xPos,'time1',time1,'airPeriod',airPeriod,'odorPeriod',odorPeriod)
flies =

    FlyName: 'fly1'
    xPos: [1x1000 double]
    time1: [1x1000 double]
    airPeriod: [1x1000 double]
    odorPeriod: [1x1000 double]
```
Using Structure Arrays

You can make a structure with `struct`:

```
>> flies = struct('FlyName','fly1','xFPos',xFPos,'time1',time1,'airPeriod',airPeriod,'odorPeriod',odorPeriod)
flies =
    FlyName: 'fly1'
    xFPos: [1x1000 double]
    time1: [1x1000 double]
    airPeriod: [1x1000 double]
    odorPeriod: [1x1000 double]
```

Access data in a structure with . notation:

```
>> flies(1).FlyName
ans =
    structure(element).fieldName
fly1
```
Using Structure Arrays

Add data to a structure using this same notation:

```
>> flies(2).FlyName = 'fly2'

flies =

1x2 struct array with fields:
    FlyName
    xPos
    time1
    airPeriod
    odorPeriod
```
Using Structure Arrays

Add data to a structure using this same notation:

```matlab
>> flies(2).FlyName = 'fly2'

flies =
1x2 struct array with fields:
    FlyName  xPos  time1  airPeriod  odorPeriod
```

Find out more about structures:
Mathematical Modeling

Mathematical modeling: describing and simulating a system with basic mathematical principles

Why build a model?
Mathematical Modeling

Mathematical modeling: describing and simulating a system with basic mathematical principles

Why build a model?

- Capture and understand the fundamental principles of your system
- Have a simpler working model that sacrifices accuracy for speed and simplicity
- Perform *in silico* experiments that would not be possible in your system
Models can be simple...

Make a dice simulator

Write a function:

- Two inputs (how many sides of the die, how many rolls)

- Inside your function, use rand to generate random rolls of the x-sided die

- Return an array filled with the generated rolls
...or not so simple...

Connectivity of a neocortical column (~10,000 neurons)
...but they all follow basic principles.

- Use basic principles to simulate the real world (often mathematical principles)
- Represent a simplification of the real world- trade accuracy for simplicity
- Often designed to capture one or two important features of a system
- Every model has inherent assumptions and limitations- know them
Stochastic Model: 1D Random Walk

We can think of a 1D random walk by moving on a number line, starting at 0:
Stochastic Model: 1D Random Walk

We can think of a 1D random walk by moving on a number line, starting at 0:

At each time point, the particle takes a step in either the positive or negative direction:

or
Stochastic Model: 1D Random Walk

Over time, there are many possible paths:

t = 1

or

or

or

or

or
How do we pick a direction?

Start at position $x=0$

At every time step:

50% chance of going left
50% chance of going right

Display position
How do we pick a direction?

Start at position x=0

At every time step:

Pick a number between 0 and 1
If the number is less than 0.5, go left
Otherwise, go right

Display position
How do we pick a direction?

Start at position x=0

At every time step:

Pick a number between 0 and 1
If the number is less than 0.5, go left
Otherwise, go right

Display position

1. type `help rand`
2. write an algorithm that does this for 100 steps
3. add `pause(0.1)` to the end of your loop
How do we pick a direction?

My solution:

```matlab
x = 0; % Set initial position to 0
figure % Create initial figure
for i = 1:100 % Loop for 100 time points
    r = rand; % Choose a random number between 0 and 1
    if r < 0.5 % Conditional statement to move left or right
        x = x-1;
    else
        x = x+1;
    end
end % Plotting: there are many ways to do this
plot(x,0,'ob','MarkerFaceColor','b'); % Plot position
% Note: 'ob' sets the markers to circles, and their color to blue
% MarkerFaceColor colors in the inside of the marker
xlim([-50 50]) % Set axis limits
pause(0.1)
```
Interpreting our results

1. Remove the plot and pause command from your script
2. Turn your script into a function that outputs the final position of the particle
3. Make a new script that runs your simulation 100 times and plots a histogram of the final particle positions

```matlab
x = 0; % Set initial position to 0
figure % Create initial figure
for i = 1:100 % Loop for 100 time points
    r = rand; % Choose a random number between 0 and 1
    if r < 0.5 % Conditional statement to move left or right
        x = x-1;
    else
        x = x+1;
    end
end
% Plotting: there are many ways to do this
plot(x,0,'ob','MarkerFaceColor','b'); % Plot position
% Note: 'ob' sets the markers to circles, and their color to blue
% MarkerFaceColor colors in the inside of the marker
xlim([-50 50]) % Set axis limits
pause(0.1)
```
Interpreting our results

1. Remove the plot and pause command from your script
2. Turn your script into a function that outputs the final position of the particle
3. Make a new script that runs your simulation 100 times and plots a histogram of the final particle positions
Interpreting our results

My solution:

```matlab
% This function performs a 1D random walk and returns the final position of % the particle after 100 time steps.

function finalPos = randomwalk

x = 0; % Set initial position to 0
for i = 1:100 % Loop for 100 time points
    r = rand; % Choose a random number between 0 and 1
    if r < 0.5 % Conditional statement to move left or right
        x = x-1;
    else
        x = x+1;
    end
end

finalPos = x;
end

positions = zeros(1,100); % Create initial positions vector
for i = 1:100 % Run the random walk 100 times
    positions(i) = randomwalk; % Store the final position in our vector
end

figure
hist(positions); % Plot a histogram of the final positions
```
Modeling biology: